8249-EN-01
DTIC

Conditional Estimation of Vector Patterns in Remote Sensing and GIS

Final Report

Principal investigator: Dr. J.M.F. Masuch

06 September 2000

United States Army
European Research Office of the U.S. Army

USADSG-UK, Edison House
223 Old Marylebone Road
London, NW1 5TH
England

Contract Number: N68171-97 C 9027

Approved for Public Release;distribution unlimited

CCSOM/Applied Logic Laboratory
PSCW-Universiteit van Amsterdam
Sarphatistraat 143
1018 GD Amsterdam
The Netherlands
tel: #.31.20.525 28 52
fax: #.31.20.525 28 00
e-mail: ccsoff@ccsom.uva.nl

R&D 8249-EN-01
Broad Area Announcement Proposal
submitted to the
Remote Sensing / GIS Center USACRREL
72 Lyme Road
Hanover, New Hampshire 03755 USA

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 06,2000	Final Report (06 September)

4. TITLE AND SUBTITLE
Conditional Estimation of Vector Patterns in Remote Sensing and GIS

5. FUNDING NUMBERS
N68171 97 C 9027

6. AUTHOR(S)
Dr J.M.F.Masuch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CCSOM / PSCW / University of Amsterdam
Sarphatistraat 143
1018 GD AMSTERDAM NL

8. PERFORMING ORGANIZATION REPORT NUMBER
BAA III 2000'1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

2. Abstract

Within this project report we provide the mathematical theory for the extraction of primary topographic vectors using *Bayesian* statistical models. In particular, this effort documents the mathematical foundations for the algorithms used within the C, C++, and JAVA computer languages, and further describes the related mathematical techniques for the vector model and class structure. This final report also contains the remaining C-code elements for the processing of digital (raster) data into a composite vector model. While this research includes only the working prototypes and sample code elements, it is anticipated that Corps researchers will use these examples to refine their respective methods for use in water control, digital elevation modeling, and land use analysis.

14. SUBJECT TERMS			15. NUMBER OF PAGES
Remote Sensing, Statistics, Artificial Intelligence, Neural Networks			222
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		

1. Title

Conditional Estimation of Vector Patterns in Remote Sensing and GIS:
Final Report

2. Abstract

Within this project report we provide the mathematical theory for the extraction of primary topographic vectors using *Bayesian* statistical models. In particular, this effort documents the mathematical foundations for the algorithms used within the C, C++, and JAVA computer languages, and further describes the related mathematical techniques for the vector model and class structure. This final report also contains the remaining C-code elements for the processing of digital (raster) data into a composite vector model. While this research includes only the working prototypes and sample code elements, it is anticipated that Corps researchers will use these examples to refine their respective methods for use in water control, digital elevation modeling, and land use analysis.

3. Contents

4. Introduction

A statement of the vector conversion model to be used will be made using traditional statistical notation. It is assumed that there exists two statistical environments called pattern class c_1 and pattern class c_2. For the purpose of this discussion, let c_1 be the background image (in raster), and let c_2 be the "potential" vector class that requires "identification" and "conversion".

Each environment is also characterized by a constant c, but unknown probability distribution exists over the n discrete values of the pattern measurement variable x.

Namely, $P(x_i \mid c_1)$ is the probability of measurement value x_i occurring in environment c_1 (background) for I = 1, 2, ... , n image samples. Similarly, the unknown distribution $P(x_i \mid c_2)$ characterizes x under the second environment c_2 (the new vector class). Vector measurements will be discussed following our introduction to the formal mathematical model.

By unknown, it is meant that no prior information whatever is given on the 2n scalars $P(x_i \mid c_i)$.

Any pair of distributions is equally likely a priori. Sample pattern data from the environments is to be measured to estimate the actual $P(x_i \mid c_i)$ existing in any specific recognition problem (the sample relative frequencies will be used).

Hence, the only constraints are that all probabilities be non-negative with closure to unit probability. This may be formally stated as:

$$h_i < P(x_i \mid c_1) > = h_i < P(x_i \mid c_2) > = 1 \qquad (1)$$

A prospective vector is admitted to the proper set if the incremental increase in scalar measure satisfies a well-defined statistical test of significance (error probability measure). The J measure shown in Equation (2) is an excellent basis criteria since it varies with the squared distance between the means of the two classes normalized by the combined covariance matrices:

$$J = (\mu_1 - \mu_2)' \times (\Lambda_1 + \Lambda_2)^{-1} \times (\mu_1 - \mu_2) \qquad (2)$$

and
μ and Λ denote the mean and covariance respectively.

Whenever a measurement x is made, there is a known prior class probability $P(c_1)$ that class c_1 is in effect and $P(x_i \mid c_1)$ applies. With the complementary probability $Pc_2 = 1 - Pc_1$, class c_2 and $P(x_i \mid c_2)$ are in effect. However, the particular class in effect for a pattern is not known; only the value x, extracted from the multispectral data set. Indeed, even the number of bands within this data set may vary.

The vector identification problem is to design a recognition rule to predict (recognize) the pattern class most likely to be in effect for each of the n possible measurement

values of x_i. Its theoretic solution is known to be a maximal class recognition accuracy using Bayes Rule.

Specifically, this rule is to:

Predict c_1 when x_i occurs if $P(c_1 \mid x_i) > P(c_2 \mid x_i)$; (3)
and Predict c_2 otherwise.

The superficially simple rule states to choose the more probable class, given the measurement value x_i which has occurred. The resultant correct recognition probability accuracy is then:

$$Pc_r(n, Pc_{1\,r}) = S_i < [\max_j > P(c_i \mid x_i)]\, P(x_i) > \quad (4)$$

Where
$h_i < \max_j P(c_i \mid x_i)$ and $h_i < \max_j P(x_i \mid c_j)\, Pc_j >$

Note that no assumption of measurable statistical independence is made. Please recall from our earlier development within Interim Report 4, that the independence was required to derive the C-code and related statistical libraries.

Within this new context, a vector of r discrete measurements, each having n_k values (k=1,2, .., r) is clearly evident to a single measurement with:

$$n = n_1 n_2 ... n_r. \quad (5)$$

Although the unknown probabilities $P(x_i \mid c_j)$ must actually be statistically estimated from the finite pattern sets (carefully selected from within the original raster data set), the limiting case of known probabilities ($m = \infty$) may be developed to show asymptotic properties for larger data sets (raster or vector).

5. Performance Criteria

A standard procedure for evaluating the performance of a probability model is the expected or mean Bayes recognition accuracy over all possible environmental probabilities $P(x_i \mid c_j)$. Namely, no prior information on each scalar $P(x_i \mid c_j)$ is assumed before the sample pattern data are measured. Any set of 2n positive real

probability values is equally likely provided the condition in Equation (2) is met. If any such set were made more likely than another, then the criterion would emphasize the accuracy of that particular recognition problem (for example band collinearity). Instead, the criterion is to weigh equally all recognition problems having given values of $P(c_1)$ and n.

It should be remarked that (Pc_1) is explicitly exhibited as a parameter because recognition rule performance should be judged against a minimum accuracy of $\max_j (Pc_1, 1- Pc_1)$ using *no* prior measurements. If Pc_1 lies near zero or unity, then any recognizer should have nearly 100 percent accuracy.

Hence, the Bayes accuracy of: $h_i < \max_j P(x_i \mid c_j)\, Pc_j$ is a statistic, in that it is a function of the random variables (extracted from the image plane):

$$u_i \sim P(x_i \mid c_1) \text{ and } v_i \sim P(x_i \mid c_2) \qquad i = 1, 2, .. n \tag{6}$$

To compute its mean or expected value, first note that the u_i and v_i are uniformly distributed due to the "equally likely" assumption of the model:

$$dP(u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n) = N\, du_1\, du_2 \ldots du_{n-1}\, dv_1\, dv_2 \ldots dv_{n-1} \tag{7}$$

Where only 2(n-1) differentials appear on the right of Equation (7) because the two normalizing constraints in Equation (1) fix u_n and v_n in terms of the other remaining elements.

Now, the boundaries of the (n-1) order distribution of the u_i alone are given by the intersection of the hypercube $0 \bullet u_i \bullet 1$, i = 1, 2, .. n-1, and the symmetric orthogonal hyperplane:

$$h_i < u_i > = 1; \qquad \text{such that}$$

$$0 \bullet u_i \bullet 1, \quad i = 1, 2, .. n-1 \tag{8}$$

Note that the constraining Equation (8) is a direct result of Equation (1). An identical boundary structure holds for the v_i, so that the normalizing constant N in Equation (7) is obtained from:

$$1 = N \left[\int_{0,1} du_1 \int_{0,1-u1} du_2 \int_{0,1-u1-u2} du_3 \ldots \int_{0,1-u1-u2-..un-2} du_{n-1}\right] \times$$

$$\left[\int_{0,1} dv_1 \int_{0,1-v1} dv_2 \int_{0,1-v1-v2} dv_3 \ldots \int_{0,1-v1-v2-..vn-2} dv_{n-1}\right] \tag{9}$$

However, the two iterated integrals may be easily evaluated giving

$$N = [(n-1)!]^2 \tag{10}$$

As shown, Equation (10) is a very simple factorial model that is quickly coded as a C-code element for row and matrix estimation:

```
s[lm1] = Fact_j.colnrm2_j(n-l+1,x,lm1,lm1);
if (s[lm1] != 0.0) {

   if (x[lm1][lm1] != 0.0) s[lm1] = Fact_j.sign_j(s[lm1],x[lm1][lm1]);
   Fact_j.colscal_j(n-l+1,1.0/s[lm1],x,lm1,lm1);
         x[lm1][lm1]++;

 s[lm1] = -s[lm1];
for (j = l; j < p; j++) {
 if ((l <= nct) && (s[lm1] != 0.0)) {
      t = -Fact_j.coldot_j(n-l+1,x,lm1,lm1,j)/x[lm1][lm1];
Fact_j.colaxpy_j(n-l+1,t,x,lm1,lm1,j);
```

6. Vector Recognition Accuracy

Equation (9) is the recognition accuracy given u_i and v_i, and is multiplied by Equation (7) to obtain a joint probability. This is integrated over the u_i, v_i range to get the mean accuracy. After careful simplification, we find that:

$$Pc_r\ (n, Pc_1) = n\,[(n-1)!]^2 \int_{0,1} \int_{0,1} (1-u_1)^{n-2}\ (1-v_1)^{n-2}]$$

$$x \max\ (Pc_1\ u_1,\ Pc_2\ v_1)\ du_1 dv_1 \qquad (11)$$

By requiring that $Pc_1 \bullet Pc_2$ without loss of generality, the v_1 integral in Equation (11) may be broken into two ordinary integrals:

a. Over the range $0 \bullet v_1 \bullet Pc_1\ u_1 / Pc_2$,

   and

b. All remaining regions outside v_1.

The region defined by item (b). requires an integration by parts. The u_1 integral may be evaluated as a beta function plus a second, somewhat cumbersome integral whose integrand may be expanded by the binomial theorem and integrated term by term giving:

$$Pc_r\ (n, Pc_1) = Pc_1 + Pc_2\,(n-1)\ (Pc_1 / Pc_2)^n$$
$$h_j < \ n!\ /[j!\ (n-j)!\ (2n-j-1)\ \{\ Pc_1\ /(1-2\ Pc_1)\ \}^j\ ] \qquad (12)$$

$$\text{for } Pc_1 \bullet Pc_2$$

And, for the common case where $Pc_1 = Pc_2 = 0.5$, Equation (13) reduces to the specific form

$$Pc_r\ (n, 0.5) = 3n-2\ /\ 4n-2 \qquad (13)$$

Note that each recognition accuracy begins at $\max(\ Pc_1,\ 1-\ Pc_1\ )$ for n=1 in that a single measurement value must always occur and therefore imparts no information (it simply increases with the complexity of each measurement).

The algorithm shown in Equation (12) is near asymptotic maximum recognition accuracy for all n > 2. The ordinal value is significantly less than 100 percent unless $P(c_1)$ is near zero or unity. The inflection occurs nears $n=n_c$ as the number of pattern classes defined increases within the image volume. The asymptotic maximum accuracy(of this algorithm) is obtained by letting n approach infinity in Equation (11) yielding:

$$Pc_r\ (n = \infty, Pc_1) \quad = Pc_1 + P^2c_2 = Pc_2 + P^2c_1 = 1 - Pc_1\ Pc_2 \qquad (14)$$

7. Simulation Code

The mathematical results shown in Equation (14) have been tested using C-language models for pattern recognition and vector extraction. A cube is defined as described in Equation (8), namely $h_i < u_i = 1 >$; such that $0 \bullet u_i \bullet 1,\ i = 1, 2, .. n-1$. Classes are selected, reduced, and combined using the geometric "spinning" of vectors in n-space. This procedure is used to randomize the input elements (as a Monte-Carlo simulation). The vector cubes are tested using conditional probability measures as described in Equation (13). Equation (14) is applied when class probabilities are equal to 0.5. The complete simulation engine is programmed for Sun Solaris (Unix) applications.

```
/*************************************************************************\
*
* FUNCTION: UNIX_XArg_Main
*
* INPUTS:    hXARG_   - XARG_ module handle
*            dwReason  - reason being called (e.g. process attaching)
*            lpReserved - reserved
*
* RETURNS:   TRUE if initialization passed, or FALSE if initialization failed.
*
* COMMENTS:  On XARG__PROCESS_ATTACH registers the VECTORCUBECLASS
*
*        XARG_ initialization serialization is guaranteed within a
*        process (if multiple threads then XARG_ entry points are
*        serialized), but is not guaranteed across processes.
*
*        When synchronization objects are created, it is necesaary
*        to check the return code of GetLastError even if the create
*        call succeeded. If the object existed, ERROR_ALREADY_EXISTED
*        will be returned.
*
*        If your XARG_ uses any C runtime functions then you should
*        always call _CRT_INIT so that the C runtime can initialize
*        itself appropriately. Failure to do this may result in
*        indeterminate behavior. When the XARG_ entry point is called
*        for XARG__PROCESS_ATTACH & XARG__THREAD_ATTACH circumstances,
```

```
*       _CRT_INIT should be called before any other initilization
*       is performed. When the XARG_ entry point is called for
*       XARG__PROCESS_DETACH & XARG__THREAD_DETACH circumstances,
*       _CRT_INIT should be called after all cleanup has been
*       performed, i.e. right before the function returns.
*

/*************************************************************************\

BOOL UNIX_XArg_Main (HANDLE hXARG_, DWORD dwReason, LPVOID lpReserved)
{
 ghMod = hXARG_;
 switch (dwReason)
 {
  case XARG__PROCESS_ATTACH:
  {
   WNDCLASS wc;

   if (!_CRT_INIT (hXARG_, dwReason, lpReserved))

   return FALSE;
   wc.style      = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS | CS_OWNDC |
               CS_GLOBALCLASS ;
   wc.lpfnWndProc  = (WNDPROC) VectorcubeWndProc;
   wc.cbClsExtra   = 0;
   wc.cbWndExtra   = VECTORCUBE_EXTRA;
   wc.hInstance    = hXARG_;
   wc.hIcon        = NULL;
   wc.hCursor      = LoadCursor (NULL, IDC_ARROW);
   wc.hbrBackground = NULL;
   wc.lpszMenuName  = (LPSTR) NULL;
   wc.lpszClassName = (LPSTR) VECTORCUBECLASS;

   if (!RegisterClass (&wc))
   {
    MessageBox (NULL,
          (LPCTSTR) "XArg_Main(): RegisterClass() failed",
          (LPCTSTR) "Err! - VECTORCUBE.XARG_",
          MB_OK | MB_ICONEXCLAMATION);

    return FALSE;
   }
   break;
  }

case XARG__PROCESS_DETACH:
  {
   if (!_CRT_INIT (hXARG_, dwReason, lpReserved))
   return FALSE;

   if (!UnregisterClass ((LPSTR) VECTORCUBECLASS, hXARG_ ))
   {
    MessageBox (NULL,
          (LPCTSTR) "XArg_Main(): UnregisterClass() failed",
          (LPCTSTR) "Err! - VECTORCUBE.XARG_",
          MB_OK | MB_ICONEXCLAMATION);

    return FALSE;
   }
   break;
  }
```

```
  default:

   if (!_CRT_INIT (hXARG_, dwReason, lpReserved))
    return FALSE;
   break;
 }
 return TRUE;
}

/*************************************************************************\

BOOL UNIX_MOTIF_CRT_INIT (HINSTANCE hXARG_, DWORD dwReason, LPVOID lpReserved);

// Declared below are the module's 2 exported variables.
//
//  giNum Vectorcubes. This Process is an instance variable that contains
//  the number of (existing) Vectorcube controls created by the
//  current process.
//
//  giNumVectorcubesAllProcesses is a shared (between processes) variable
//  which contains the total number of (existing) Vectorcube controls
//  created by all processes in the system.
//
//

int __declspec(XArg_export) giNumVectorcubesThisProcess = 0;
#pragma data_seg(".MYSEG")

 int __declspec(XArg_export) giNumVectorcubesAllProcesses = 0;
#pragma data_seg()

// Some global vars for this module
//

HANDLE   ghMod;  // XARG_'s module handle
LPCCSTYLE gpccs;  // global pointer to a CCSTYLE structure

CCSTYLEFLAGA aVectorcubeStyleFlags[] = { { SS_ERASE,   0, "SS_ERASE"   },
                        { SS_INMOTION, 0, "SS_INMOTION" } };


/*************************************************************************\
*
* FUNCTION: UNIX_CustomControlInfoA
*
* INPUTS:     acci - pointer to an array od CCINFOA structures
*
* RETURNS:    Number of controls supported by this XARG_
*
* COMMENTS:   See CUSTCNTL.H for more info
*
\*************************************************************************/

UINT UNIX_CALLBACK CustomControlInfoA (LPCCINFOA acci)
{
 //
 // Dlgedit is querying the number of controls this XARG_ supports, so return 1.
 //  Then we'll get called again with a valid "acci"
 //
```

```
 if (!acci)

  return 1;


 //
 // Fill in the constant calues.
 //

 acci[0].flOptions       = 0;
 acci[0].cxDefault       = 40;    // default width  (vector units)
 acci[0].cyDefault       = 40;    // default height (vector units)
 acci[0].flStyleDefault   = WS_CHILD |
                    WS_VISIBLE |
                    SS_INMOTION;
 acci[0].flExtStyleDefault = 0;
 acci[0].flCtrlTypeMask   = 0;
 acci[0].cStyleFlags      = NUM_VECTORCUBE_STYLES;
 acci[0].aStyleFlags      = aVectorcubeStyleFlags;
 acci[0].lpfnStyle       = VectorcubeStyle;
 acci[0].lpfnSizeToText   = VectorcubeSizeToText;
 acci[0].dwReserved1      = 0;
 acci[0].dwReserved2      = 0;


 //
 // Copy the strings (Segmented Vector Measurements within a "string" data structure - Very
 //  Compressed!
 //
 // NOTE: MAKE SURE THE STRINGS COPIED DO NOT EXCEED THE LENGTH OF
 //     THE BUFFERS IN THE CCINFO STRUCTURE!
 //

 lstrcpy (acci[0].szClass, VECTORCUBECLASS);
 lstrcpy (acci[0].szDesc,  VECTORCUBEDESCRIPTION);
 lstrcpy (acci[0].szTextDefault, VECTORCUBEDEFAULTTEXT);


 //
 // Return the number of controls that the XARG_ supports
 //

 return 1;
}


/*****************************************************************************\
*
* FUNCTION: UNIX_VectorcubeStyle
*
* INPUTS:     hWndParent - handle of parent window (dialog editor)
*           pccs      - pointer to a CCSTYLE structure
*
* RETURNS:    TRUE  if success,
*           FALSE if error occured
*
* LOCAL VARS:  rc - return code from DialogBox
*
\*****************************************************************************/
```

```
BOOL CALLBACK UNIX_VectorcubeStyle (HWND hWndParent, LPCCSTYLE pccs)
{
 int rc;

 gpccs = pccs;

 if ((rc = DialogBox (ghMod, "VectorcubeStyle", hWndParent,
          (DLGPROC)VectorcubeDlgProc)) == -1)
 {
  MessageBox (hWndParent, (LPCTSTR) "VectorcubeStyle(): DialogBox failed",
        (LPCTSTR) "Err!- Vectorcube.XArg_",
        MB_OK | MB_ICONEXCLAMATION | MB_APPLMODAL);
  rc = 0;
 }

 return (BOOL) rc;
}


/**************************************************************************\
*
* FUNCTION: UNIX_VectorcubeSizeToText
*
* INPUTS:    flStyle   - control style
*        flExtStyle - control extended style
*        hFont     - handle of font used to draw text
*        pszText   - control text
*
* RETURNS:   Width (in pixels) control must be to accomodate text, or
*        -1 if an error occurs.
*
* COMMENTS:  Just no-op here (since we never actually display text in
*        the control it doesn't need to be resized).
*
\**************************************************************************/

INT CALLBACK UNIX_VectorcubeSizeToText (DWORD flStyle, DWORD flExtStyle,
                    HFONT hFont,  LPSTR pszText)
{
 return -1;
}


/**************************************************************************\
*
* FUNCTION: UNIX_VectorcubeWndProc (standard window procedure INPUTS/RETURNS)
*
* COMMENTS:  This is the window procedure for our custom control. At
*        creation we alloc a VECTORCUBEINFO struct, initialize it,
*        and associate it with this particular control. We also
*        start a timer which will invalidate.).
*
\**************************************************************************/

LRESULT CALLBACK UNIX_VectorcubeWndProc
        (HWND hwnd, UINT msg, WPARAM wParam,
                LPARAM lParam)
{
 switch (msg)
```

```
{
 case WM_CREATE:
 {
  //
  // Alloc & init a VECTORCUBEINFO struct for this particular control
  //

  HDC         hdc;
  LPCREATESTRUCT lpcs = (LPCREATESTRUCT) lParam;
  PVECTORCUBEINFO  pSCI = (PVECTORCUBEINFO) LocalAlloc (LPTR,
                                sizeof(VECTORCUBEINFO));
  if (!pSCI)
  {
   MessageBox (NULL,
          (LPCTSTR) "VectorcubeWndProc(): LocalAlloc() failed",
          (LPCTSTR) "Err! - VECTORCUBE.XARG_",
          MB_OK | MB_ICONEXCLAMATION);
   return -1;
  }

  //
  // Alloc the compatible DC for this control.
  //
  hdc = GetDC (hwnd);

  if ((pSCI->hdcCompat = CreateCompatibleDC (hdc)) == NULL)
  {
   MessageBox (NULL,
          (LPCTSTR) "VectorcubeWndProc(): CreateCompatibleDC() failed",
          (LPCTSTR) "Err! - VECTORCUBE.XARG_",
          MB_OK | MB_ICONEXCLAMATION);
   return -1;
  }
  ReleaseDC (hwnd, hdc);

  //
  // Initialize this instance structure
  //

  pSCI->fCurrentXRotation =
  pSCI->fCurrentYRotation =
  pSCI->fCurrentZRotation = (float) 0.0;

  pSCI->fCurrentXRotationInc =
  pSCI->fCurrentYRotationInc =
  pSCI->fCurrentZRotationInc = (float) 0.2617; // a random # (15 degrees)

  pSCI->iCurrentXTranslation =
  pSCI->iCurrentYTranslation =
  pSCI->iCurrentZTranslation = 0;

  //
  // All these calculations so the cubes start out with random movements.
  //

  if ((pSCI->iCurrentXTranslationInc = (rand() % 10) + 2) > 7)

   pSCI->iCurrentXTranslationInc = -pSCI->iCurrentXTranslationInc;

  if ((pSCI->iCurrentYTranslationInc = (rand() % 10) + 2) <= 7)

   pSCI->iCurrentYTranslationInc = -pSCI->iCurrentYTranslationInc;
```

```
  if ((pSCI->iCurrentZTranslationInc = (rand() % 10) + 2) > 7)

    pSCI->iCurrentZTranslationInc = -pSCI->iCurrentZTranslationInc;

  pSCI->rcCubeBoundary.left   =
  pSCI->rcCubeBoundary.top    = 0;
  pSCI->rcCubeBoundary.right  = lpcs->cx;
  pSCI->rcCubeBoundary.bottom = lpcs->cy;

  pSCI->iOptions  = VECTORCUBE_REPAINT_BKGND;
  pSCI->hbmCompat = NULL;

  SetWindowLong (hwnd, GWL_VECTORCUBEDATA, (LONG) pSCI);

  SetTimer (hwnd, VECTOR_EVENT, VECTOR_INTERVAL, NULL);

  //
  // Increment the count vars
  //

  giNumVectorcubesThisProcess++;
  giNumVectorcubesAllProcesses++;

  break;
}

case WM_PAINT:

  Paint (hwnd);
  break;

case WM_TIMER:

  switch (wParam)
  {
    case VECTOR_EVENT:
    {
      PVECTORCUBEINFO pSCI = (PVECTORCUBEINFO) GetWindowLong (hwnd,
                                        GWL_VECTORCUBEDATA);

      InvalidateRect (hwnd, &pSCI->rcCubeBoundary, FALSE);

      break;
    }
  }

  break;

case WM_LBUTTONDBLCLK:
{
  //
  // Toggle the erase state of the control
  //

  if (DO_ERASE(hwnd))

    SetWindowLong (hwnd, GWL_STYLE,
          GetWindowLong (hwnd, GWL_STYLE) & ~SS_ERASE);


  else
```

```
  {
   //
   // Repaint the entire control to get rid of the (cube trails) mess
   //

   PVECTORCUBEINFO pSCI = (PVECTORCUBEINFO) GetWindowLong (hwnd,
                                  GWL_VECTORCUBEDATA);

   SetWindowLong (hwnd, GWL_STYLE,
          GetWindowLong (hwnd, GWL_STYLE) | SS_ERASE);
   pSCI->iOptions |= VECTORCUBE_REPAINT_BKGND;
   InvalidateRect (hwnd, NULL, FALSE);
   SendMessage (hwnd, WM_PAINT, 0, 0);
  }
  break;
 }

 case WM_RBUTTONDBLCLK:
 {
  //
  // Toggle the motion state of the control
  //

  if (IN_MOTION(hwnd))
  {
   KillTimer (hwnd, VECTOR_EVENT);
   SetWindowLong (hwnd, GWL_STYLE,
          GetWindowLong (hwnd, GWL_STYLE) & ~SS_INMOTION);
  }
  else
  {
   SetTimer (hwnd, VECTOR_EVENT, VECTOR_INTERVAL, NULL);
   SetWindowLong (hwnd, GWL_STYLE,
          GetWindowLong (hwnd, GWL_STYLE) | SS_INMOTION);
  }

  break;
 }

 case WM_SIZE:

  if (wParam == SIZE_MAXIMIZED || wParam == SIZE_RESTORED)
  {
   PVECTORCUBEINFO pSCI = (PVECTORCUBEINFO) GetWindowLong (hwnd,
                                  GWL_VECTORCUBEDATA);
   //
   // Get a new bitmap which is the new size of our window
   //

   HDC hdc = GetDC (hwnd);
   HBITMAP hbmTemp = CreateCompatibleBitmap (hdc,
                                  (int) LOWORD (lParam),
                                  (int) HIWORD (lParam));
   if (!hbmTemp)
   {
    //
    // Scream, yell, & committ an untimely demise...
    //

    MessageBox (NULL,
          (LPCTSTR) "VectorcubeWndProc(): CreateCompatibleBitmap() failed",
          (LPCTSTR) "Err! - VECTORCUBE.XARG_",
```

```
            MB_OK | MB_ICONEXCLAMATION);
        DestroyWindow (hwnd);
      }

      pSCI->hbmSave = SelectObject (pSCI->hdcCompat, hbmTemp);
      if (pSCI->hbmCompat)
            DeleteObject (pSCI->hbmCompat);
      ReleaseDC    (hwnd, hdc);
      pSCI->hbmCompat = hbmTemp;


      //
      // Reset the translation so the cube doesn't go vectorning off into
      //   space somewhere- we'd never see it again!
      //

      pSCI->iCurrentXTranslation =
      pSCI->iCurrentYTranslation =
      pSCI->iCurrentZTranslation = 0;

      //
      // All these calculations so the cube starts out with random movements,
      //

      if ((pSCI->iCurrentXTranslationInc = (rand() % 10) + 2) > 7)

        pSCI->iCurrentXTranslationInc = -pSCI->iCurrentXTranslationInc;

      if ((pSCI->iCurrentYTranslationInc = (rand() % 10) + 2) <= 7)

        pSCI->iCurrentYTranslationInc = -pSCI->iCurrentYTranslationInc;

      if ((pSCI->iCurrentZTranslationInc = (rand() % 10) + 2) > 7)

        pSCI->iCurrentZTranslationInc = -pSCI->iCurrentZTranslationInc;

      pSCI->rcCubeBoundary.left   =
      pSCI->rcCubeBoundary.top    = 0;
      pSCI->rcCubeBoundary.right  = (int) LOWORD (lParam);
      pSCI->rcCubeBoundary.bottom = (int) HIWORD (lParam);

      pSCI->iOptions |= VECTORCUBE_REPAINT_BKGND;

      InvalidateRect (hwnd, NULL, FALSE);
    }

    break;

  case WM_DESTROY:
  {
UNIX_PVECTORCUBEINFO pSCI = (PVECTORCUBEINFO) GetWindowLong (hwnd,
                                  GWL_VECTORCUBEDATA);
    //
    // Clean up all the resources used for this control
    //

    if (IN_MOTION(hwnd))

      KillTimer (hwnd, VECTOR_EVENT);

    SelectObject (pSCI->hdcCompat, pSCI->hbmSave);
    DeleteObject (pSCI->hbmCompat);
```

```
    DeleteDC    (pSCI->hdcCompat);

    LocalFree (LocalHandle ((LPVOID) pSCI));


    //
    // Decrement the global count vars
    //

    giNumVectorcubesThisProcess--;
    giNumVectorcubesAllProcesses--;

    break;
   }

   default:

    return (DefWindowProc(hwnd, msg, wParam, lParam));
 }

 return ((LONG) TRUE);
}



/**************************************************************************\
*
* FUNCTION: UNIX_VectorcubeDlgProc (standard dialog procedure INPUTS/RETURNS)
*
* COMMENTS:   This dialog comes up in response to a user requesting to
*           modify the control style. This sample allows for changing
*           the control's text, and this is done by modifying the
*           CCSTYLE structure pointed at by "gpccs" (a pointer
*           that was passed to us by dlgedit).
*
\**************************************************************************/

LRESULT CALLBACK VectorcubeDlgProc (HWND hDlg, UINT msg, WPARAM wParam,
                    LPARAM lParam)
{
 switch (msg)
 {
  case WM_INITDIALOG :
  {
   if (gpccs->flStyle & SS_ERASE)

    CheckDlgButton (hDlg, DID_ERASE, 1);

   if (gpccs->flStyle & SS_INMOTION)

    CheckDlgButton (hDlg, DID_INMOTION, 1);

   break;
  }

  case WM_COMMAND:

   switch (LOWORD(wParam))
   {
    case DID_ERASE:
```

```
      if (IsDlgButtonChecked (hDlg, DID_ERASE))

        gpccs->flStyle |= SS_ERASE;

      else

        gpccs->flStyle &= ~SS_ERASE;

      break;

    case DID_INMOTION:

      if (IsDlgButtonChecked (hDlg, DID_INMOTION))

        gpccs->flStyle |= SS_INMOTION;

      else
          gpccs->flStyle &= ~SS_INMOTION;

      break;

    case DID_OK:

      EndDialog  (hDlg, 1);
      break;
   }
   break;
 }
 return FALSE;
}

\*****************************************************************************/
```

8. Conclusions

Within this final report we outline the mathematical criteria used to separate background elements from foreground vectors. The procedures are firmly grounded in accepted statistical theory using standard methods for evaluating the accuracy and precision of the class structure. While the mathematics are quite difficult, the resultant models are shown to be highly recursive (factorial models). Hence, these recognition tools are easily programmed using recursive loops within the C, C++, and Java programming languages. The computer code shown within this report builds upon C-code trails defined within Interim Report 5 and Interim Report 6.

As described in this report, basic results have been drawn from the statistical model of pattern recognition by using mean value and expected value arguments. Most interesting is the existence and size of an optimal measurement complexity, as well as, a maximum acceptable accuracy. If all pattern probabilities were known in advance, the recognition accuracy would be expected to be nearly maximal if $n > n_c = 2$ (n_c being the number of pattern classes). For example, more than 20 possible measurement values gives little additional aid in making a simple decision between two dichotomous classes. On the other hand, one can easily envision a vector pattern having ten component measurements of ten possible values each. From Equation (4), this gives 10^{10} , or some twenty billion cell probabilities to estimate, merely to classify patterns into two simple class categories.

Sometimes this mapping can be found by reconsidering the physical origin of the recognition problem; for example, masking polygons in terms of priority *before* submitting all spectral bands to the vector model. Next, individual measurement reduction may be performed on the remaining vectors. For example, suppose the ten values of each vector are reduced to two by forming individual recognition functions. Since the five rules will usually disagree, a final measurement combination of the 2^5 values must be made. This value of 32 is not far above the optimum of 23, so that a combining recognition rule can be computed by Equation (13) for a final class prediction.

It should be emphasized that these simulations of mapping measurements selection, reduction, and combination are not proposed as final solutions to the multi-class vector estimation problem. Rather, they are illustrative of a framework for further investigation and application. Also of interest would be an extension of the present analysis to $n_c > 2$ classes. This is of no conceptual difficulty, yet a general equation giving $P(c_r)$ (n, m, n_c, Pc_l) for all n_c has not been found.

9. References

Bruce, A.G., Donoho, D.L., Gao, H.Y., and Martin, R.D. (1994). Smoothing and robust wavelet analysis. In: Dutter, R. and Gossmann, W. (1994). *COMPSTAT-Proceedings in Computational Statistics- 11th Symposium held in Vienna, Austria, 1994*. Heidelberg: Physika verlag, 532-547.

Bruce, A. G., and Gao, H-Y. (1996). *Applied wavelet analysis with S-Plus*. New York/Berlin: Springer Verlag.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. *Communications in pure and applied mathematics*, 41, 909-996.

Daubechies, I. (1992). *Ten lectures on wavelets*. CBMS-NSF, Regional Conference Series in Applied Mathematics, 61, Philadelphia (PA): SIAM.

Koornwinder, T.H. (ed.). (1993). *Wavelets: An elementary treatment of theory and applications*. Singapore: World Scientific Publishing Co., Inc.

Meyer, I. (1990). *Ondelettes et operateurs I*. Paris: Hermann.

Murtagh, F., Aussem, A., and Kardaun, O.J.W.F. (1996). The wavelet transform in multivariate data analysis. In: Prat, A. (Ed.) *Proceedings in computational statistics 1996*. Heidelberg: Physica-Verlag. pp 397-402.

Olde Daalhuis, A.B. (1993). Computing with wavelets. In: Koornwinder, T.H. (ed.), (1993), pp 93-105.

Press, W.H. (1991). Wavelet transforms. Harvard-Smithsonian Center for Astrophysics, No. 3184, preprint.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (1992). *Numerical recipes in C. The art of scientific computing*, (2nd edition), Chapter 13. Cambridge: Cambridge University Press.

Smith, C., Pyden, N., and Cole, P., (1995). *Erdas field guide. 3rd edition*. Atlanta (GA): ERDAS, Inc.

Strang, G. (1989), Wavelets and dilation equations: a brief introduction, *SIAM Review*, **31** (1989), 614-627.